1. Refined insights

We're using the data that was obtained at the end of Milestone 1.

- Data contains users who listened to very little songs and also songs that have been listened only few times, we're going to drop those records, saving only users who listened to at **least 90 or more** songs and songs that have been listened by **120 or more** users in our final data frame.
- We're also using only data (records) with max play_count 5 (play_count <= 5).
- Final data frame has **3,155** unique users, **563** unique songs and **232** unique artists.
- We have 3,155 * 563 = **1,776,265** possible interactions between users and songs.
- We already have 117,876 interactions in our final data frame, so we have 1,776,265 - 117,876 = **1,658,389** possible interactions left.
- Since we don't have an actual rating, we'll use *play_count* as a rating with max rating/play_count **5** and min *play_count* **0**.
- We're setting up **threshold** at **1.5**, meaning if average *play_count* for a given song is >=1.5, then we'll recommend this song to a user, if it's <1.5, then we won't recommend it.
- Here's a bar plot for *play_count*

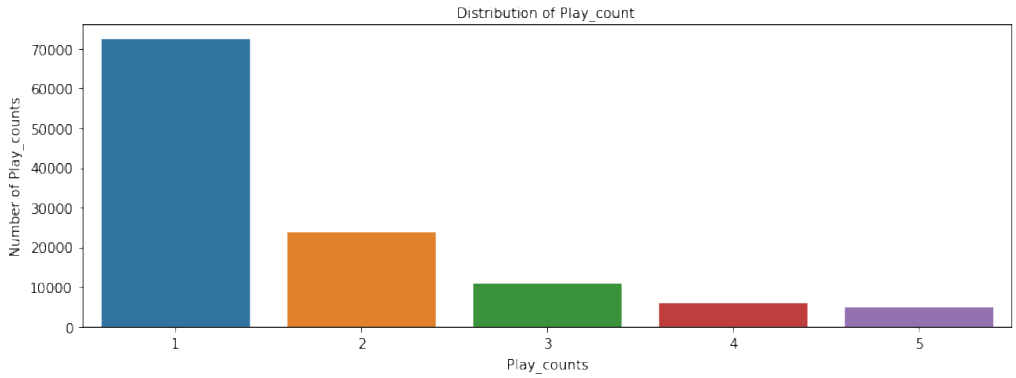

Figure 1: Distribution of Play_count.

- As we can see, the most common value is 1 (~700K), then 2 and then 3, 4 and 5.
- We're creating a data frame and calculating average play_count per song (avg_count).
- We can see user-song interaction distribution

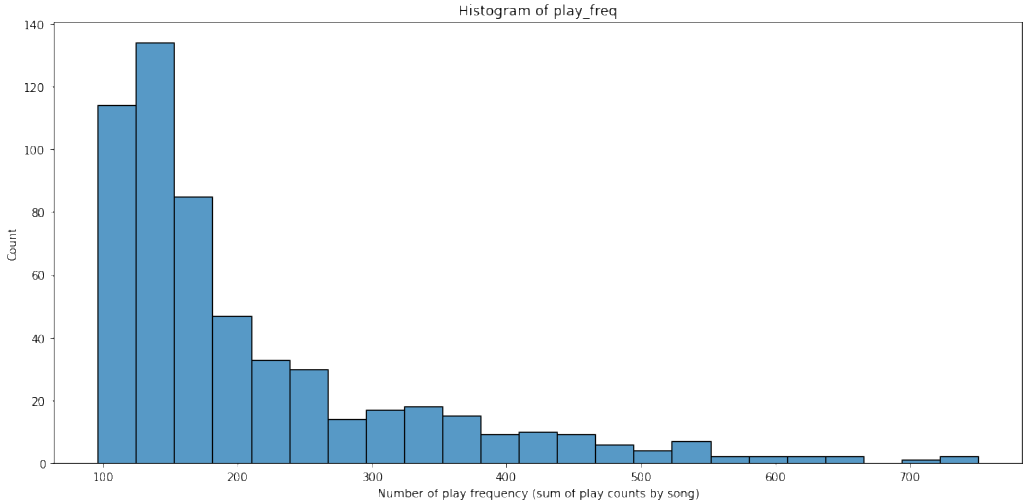

Figure 2: Histogram of Play_freq.

- The distribution is highly skewed to the right.
- It clearly shows that there are very few songs which have been played many times.

- Rank-based recommendation systems provide recommendations based on the most popular items (songs). This kind of recommendation system is useful when we have cold start problems. Cold start refers to the issue when we get a new user into the system and the machine is not able to recommend song to the new user, as the user did not have any historical interactions in the dataset. In those cases, we can use **rank-based recommendation system** to recommend song to the new user.

- To build the **rank-based recommendation system**, we take average of all the ratings/play_count provided to each song and then rank them based on their average rating / play_count.

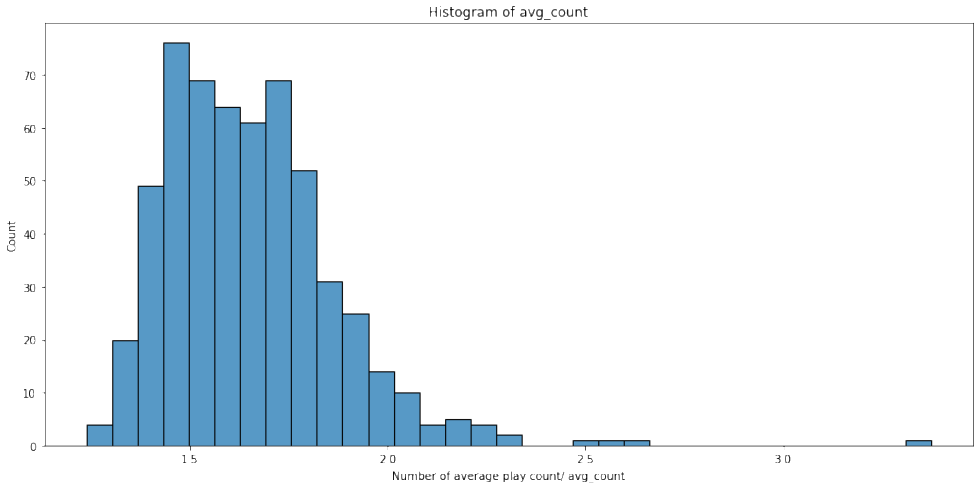

Figure 3: Histogram of avg_count.

- I've created a function to find the top n songs for a recommendation based on the average play count of song. I also added a threshold for a minimum number of *play_counts* for a song to be considered for recommendation (**threshold =1.5**).

Terminology used in the report (and Python code):

- **Relevant item** - An item (song in this case) that is actually **rated higher than the threshold rating (here 1.5)** is relevant, if the **actual rating is below the threshold then it is a non-relevant item**.

- **Recommended item** - An item that's **predicted rating is higher than the threshold (here 1.5) is a recommended item**, if the **predicted rating (avg_count) is below the threshold then that song will not be recommended to a user**.
- **False Negative (FN)** - It is the **frequency of relevant items that are not recommended to the user**. If the relevant items are not recommended to the user, then the user might not listen to the song. This would result in the **loss of opportunity for the service provider** which they would like to minimize.

- **False Positive (FP)** - It is the **frequency of recommended items (songs) that are actually not relevant**. In this case, the recommendation system is not doing a good job of finding and recommending the relevant items to the user. This would result in **loss of resources for the service provider** which they would also like to minimize.

- **Precision** - It is the **fraction of recommended items (songs) that are relevant actually** i.e. if out of 10 recommended items, 6 are found relevant by the user then precision is 0.60. The higher the value of precision better is the model. It is one of the metrics to do the performance assessment of classification models.
- **Recall** - It is the **fraction of actually relevant items (songs) that are recommended to the user** i.e., if out of 10 relevant books, 6 are recommended to the user then recall is 0.60. Higher the value of recall better is the model. It is one of the metrics to do the performance assessment of classification models.
- **RMSE** – Root Mean Square Error

2. Comparison of various techniques and their relative performance

Here are the techniques I've implemented

1. Popularity-based Recommendation System
2. Similarity Based Collaborative Filtering (user- user)
3. Similarity Based Collaborative Filtering (item-item)
4. Matrix Factorization
5. Cluster Based Recommendation System
6. Content Based Recommendation System

Perf eval measures	Similarity Based Collaborative Filtering				Model Based Collaborative Filtering		Cluster based Recommendation System	
	User - user		Item - item		Matrix Factorization			
	Baseline	Optimized	Baseline	Optimized	Baseline	Optimized	Baseline	Optimized
RMSE	1.0878	1.0521	1.0394	1.0328	1.0252	**1.0141**	1.0691	1.0487
Precision	0.396	0.413	0.307	0.408	0.41	**0.415**	0.398	0.397
Recall	0.692	**0.721**	0.562	0.665	0.633	0.635	0.56	0.582
F1	0.504	0.525	0.397	0.506	0.498	0.502	0.465	0.472

User_id	Song_id								
6958	1671	Pred1	1.801	**1.963**	1.361	1.963	1.267	1.343	1.294	1.294
6958	3232	Pred2	1.639	1.452	1.378	1.492	1.556	1.804	1.479	1.479
27018	1671	Pred3	1.275	1.286	2.551	2.337	2.217	2.264	2.403	2.403

Table 1: Model comparison.

Table above presents performance evaluation measures for few models (different algorithms, baseline model and optimized models – model after tuning hyperparameters).
It also shows predictions for a certain user and song, first one is for a known user-item interaction (actual value is 2) and last two are predictions for an unknows user-item interaction.

- As we can see from the table, the optimized models performed much better than the baseline ones.
- For all of the models Recall is much higher than Precision, meaning we're much better at recommending relevant songs than choosing relevant songs to be recommended (60-70% of songs that are recommended aren't relevant).
- This can cost us a lot.
- Two models that got my attention are **User-User Optimized Model for Similarity Based Collaborative Filtering** and **Optimized Model for Collaborative Filtering Matrix Optimization**.
- **Item-item Optimized Model for Similarity Based Collaborative Filtering** also performs well.

- Looks like **Optimized Matrix Factorization** model has the **lowest RMSE** which means that the model fits data very well.
- This model has also the best (highest) **Precision** (fraction of recommended items that are relevant actually) of all of the models.
- However, its **Recall** (fraction of actually relevant songs that are recommended to the user) isn't the best, even though they're pretty close to the best ones.
- Another good model is **User-User Optimized Model for Similarity Based Collaborative Filtering** with best Recall of all of the researched models.

- We can also see an example prediction for a given user and song.
- It is interesting to note that depending on the model, we would either recommended a certain song to a user or not.
- Some models won't even recommend a song even though it's relevant (predicted value of *avg_count* < 1.5 while actual value = 2)

- I am not discussing F1 score since it's a linear combination of Precision and Recall.
- I also looked at recommendations for each model and its corrected ratings.
- Interesting enough, both models (User-user and Matrix Factorization) recommended different songs for the same user, however some of those songs were also recommended based on popularity-based model and cluster model.
- Full list of recommendations is provided in Jupyter Python Notebook.

- For further improvements of the model, we can take a closer look into data preparation and more hyper parameters tuning.
- I would also investigate different values of threshold.
- I've noticed that choosing lower threshold, we might get better values of Precision and Recall (so also F1 Score), but then the model wouldn't make much sense.

3. Proposal for the final solution design
    - My proposition would be to adopt **User-User Optimized Model for Similarity Based Collaborative Filtering.**
    - This solution provides the best output in terms of evaluation values **Recall** and its **Precision** is just slightly worse performing than the other best model (**Optimized Model for Collaborative Filtering Matrix Optimization**)
    - From the evaluation perspective, those two models perform in a very similar way.
    - **User-User Optimized Model for Similarity Based Collaborative Filtering** recommends more unique songs (meaning other models' predictions didn't overlap much) with higher corrected predicted ratings.
    - Fact that matrix factorization's predictions overlap with other predictions could lead us to use this model instead, however I'll stick with **User-User Optimized Model for Similarity Based Collaborative Filtering.**